Interpretation methods in ICC

Jean-Yves Moyen
Jean-Yves.Moyen@lipn.univ-paris13.fr

Université Paris 13 (LIPN)

November 2013

Introduction

What this talk is about

- What this is not about:
  Collection of all results of ICC using interpretations.

What this talk is about

- What this is not about:
  Collection of all results of ICC using interpretations.
- What this is (probably) about:
  - Tentative definition of "ICC".
  - From termination orderings to interpretations.
  - How interpretations help in ICC.

Implicit Computational Complexity

Implicit Computational Complexity

Decidable syntactic criterions for semantics properties.

Implicit Computational Complexity

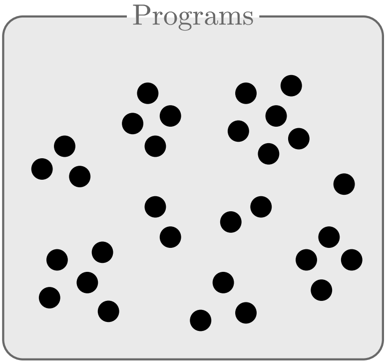
Decidable syntactic criterions for semantics properties.

Set of programs

Set of functions

Implicit Computational Complexity

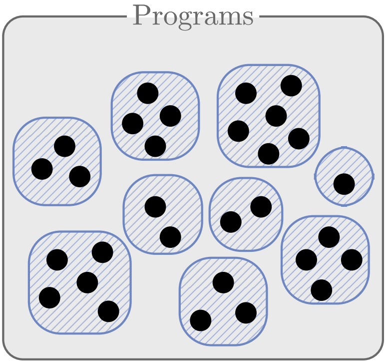
Decidable syntactic criterions for semantics properties.

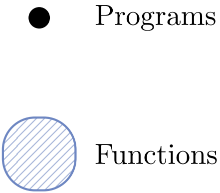


- Programs

Implicit Computational Complexity

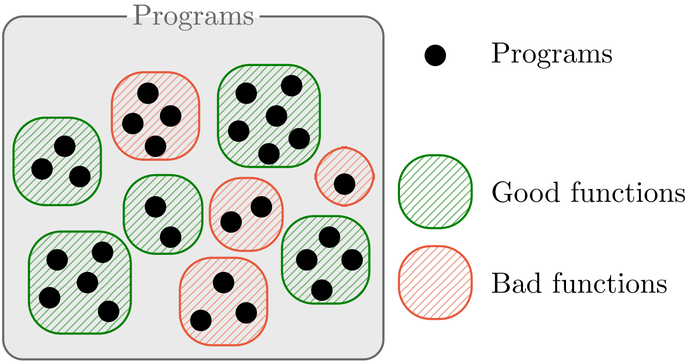
Decidable syntactic criterions for semantics properties.





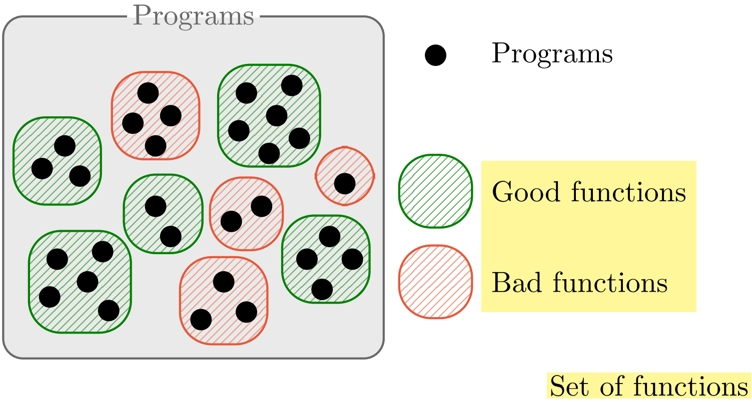
Implicit Computational Complexity

Decidable syntactic criterions for semantics properties.



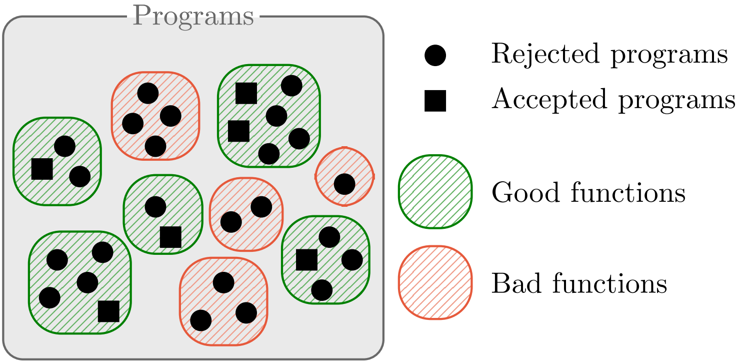
Implicit Computational Complexity

Decidable syntactic criterions for semantics properties.



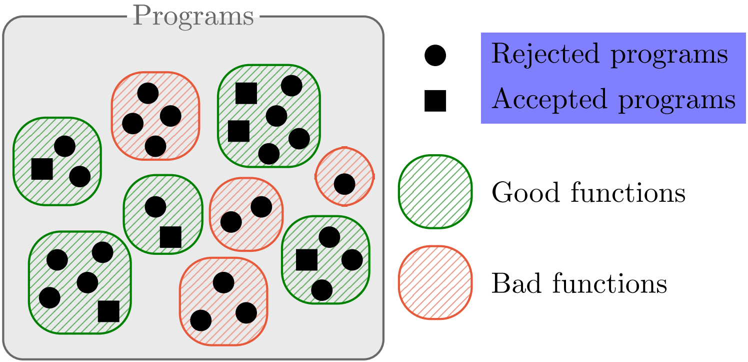
Implicit Computational Complexity

Decidable syntactic criterions for semantics properties.



Implicit Computational Complexity

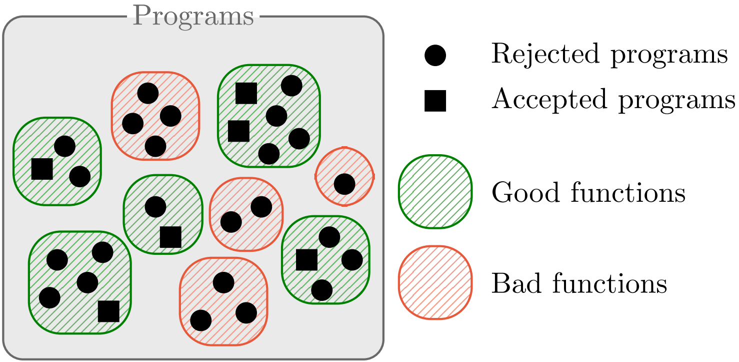
Decidable syntactic criterions for semantics properties.

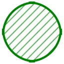
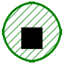


Set of programs

Implicit Computational Complexity

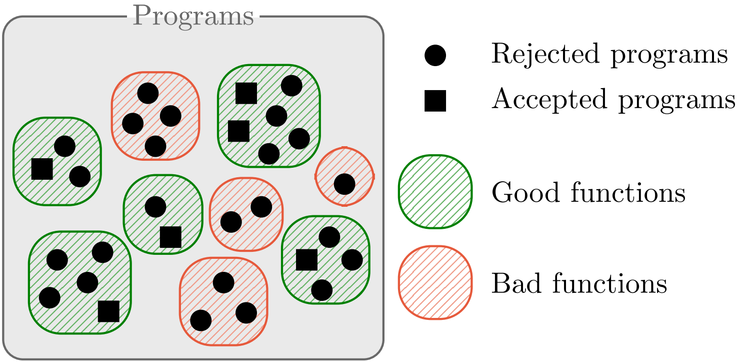
Decidable syntactic criterions for semantics properties.

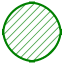
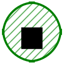


- For each , there exists .

Implicit Computational Complexity

Decidable syntactic criterions for semantics properties.



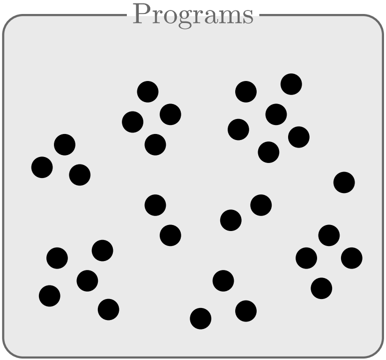
- For each , there exists .
- There is no .

Time Complexity

A function is PTIME iff it is computed by at least one polytime program.

Time Complexity

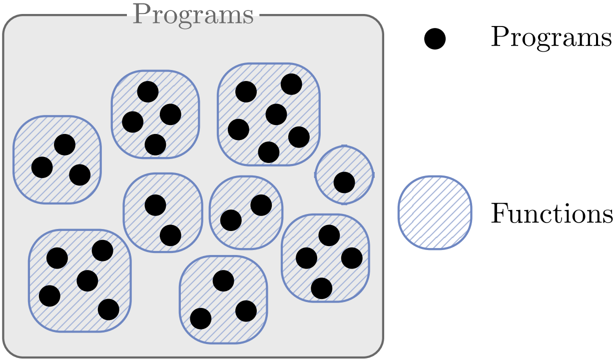
A function is PTIME iff it is computed by at least one polytime program.



● Programs

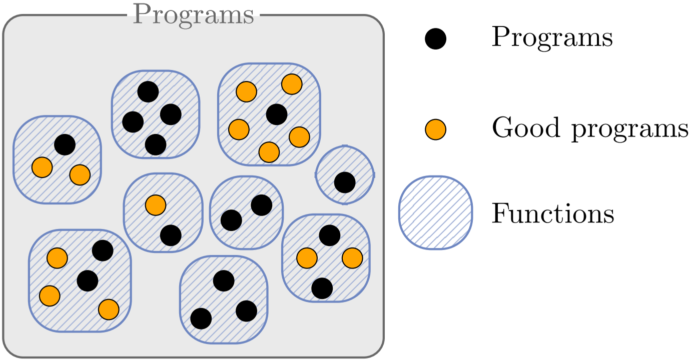
Time Complexity

A function is PTIME iff it is computed by at least one polytime program.



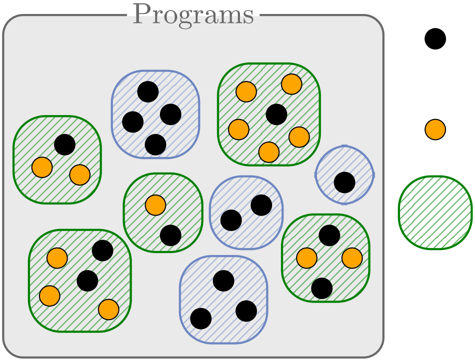
Time Complexity

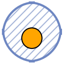
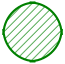
A function is PTIME iff it is computed by at least one polytime program.



Time Complexity

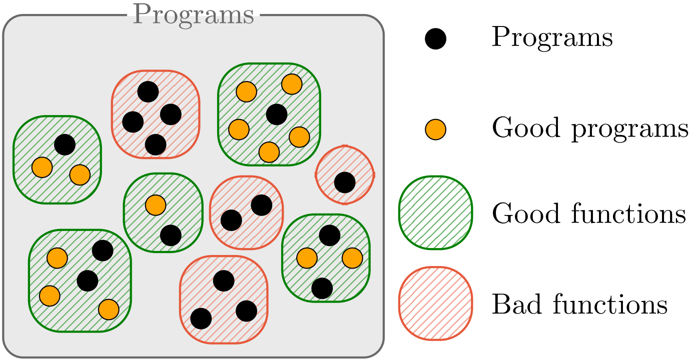
A function is PTIME iff it is computed by at least one polytime program.



If there is , then .

Time Complexity

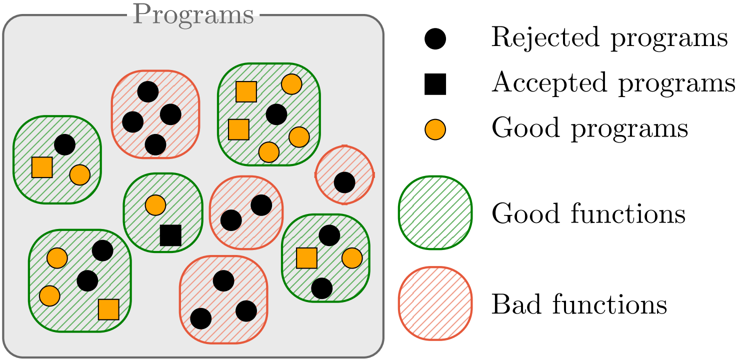
A function is PTIME iff it is computed by at least one polytime program.



If there is no , then .

Time Complexity

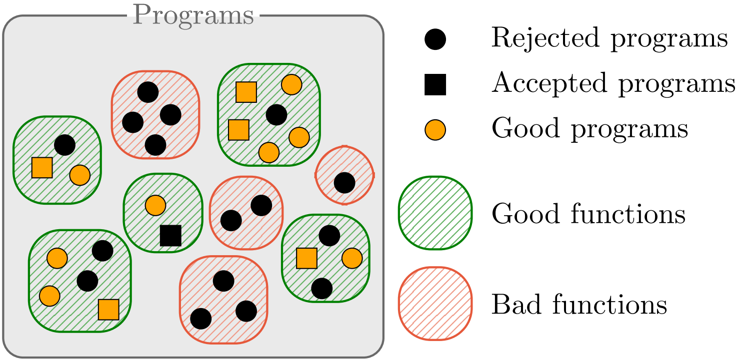
A function is PTIME iff it is computed by at least one polytime program.



At least one ■ witness for each .

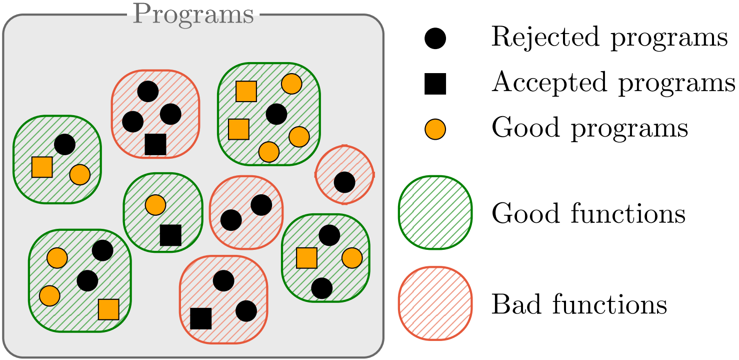
Properties of ICC Systems

Properties of ICC Systems



	F. pos.	F. neg.
W		
S		

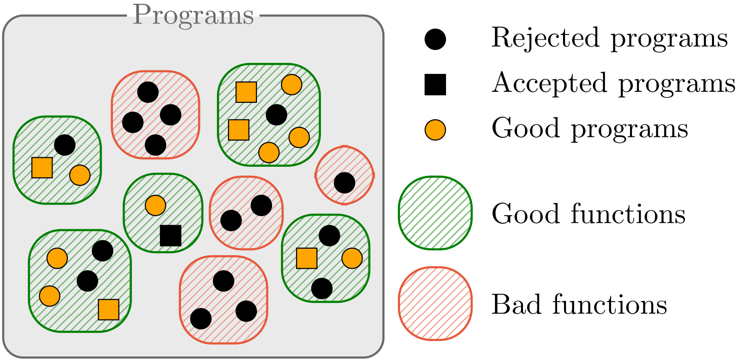
Properties of ICC Systems



	F. pos.	F. neg.
W	No	
S		

Soundness

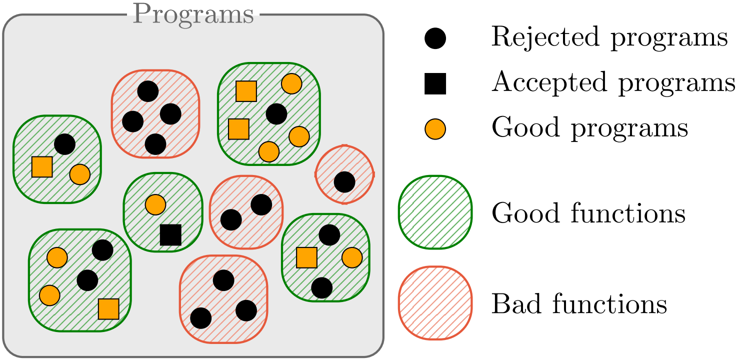
Properties of ICC Systems



	F. pos.	F. neg.
W	No	
S	No	

Soundness

Properties of ICC Systems

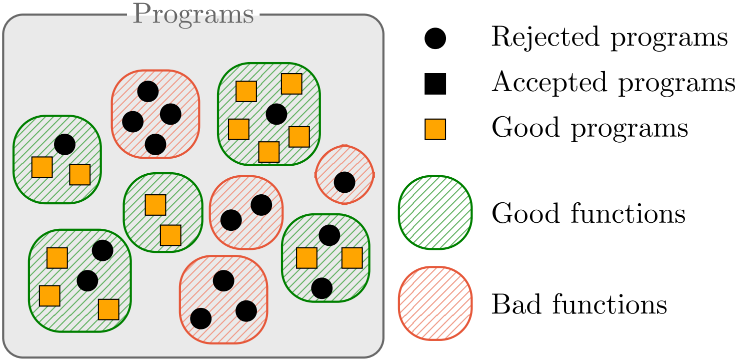


	F. pos.	F. neg.
W	No	
S	No	

Soundness

Hard to prove

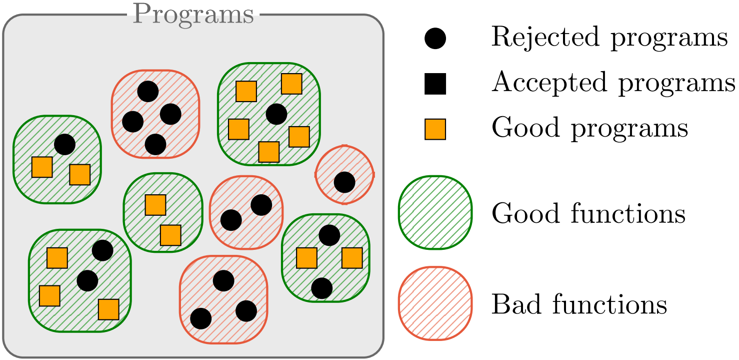
Properties of ICC Systems



	F. pos.	F. neg.
W	No	
S	No	No

Intensional completeness

Properties of ICC Systems

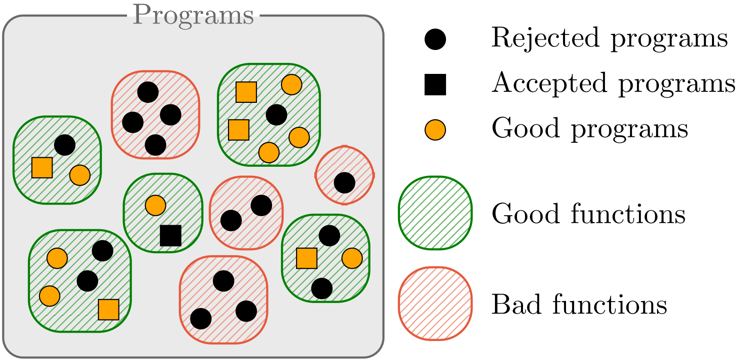


	F. pos.	F. neg.
W	No	
S	No	No

Intensional completeness

Undecidable!

Properties of ICC Systems



	F. pos.	F. neg.
W	No ■	Some ■ in each
S	No ■	No

Extensional completeness

Examples

- A program without loops always terminate.

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.

Examples

Syntactic criterion

Semantic property

- A program without loops $\boxed{\text{always}}$ terminate.

$\boxed{\text{Sound}}$

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.

  Some total functions need loops.

Sound

Incomplete

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.
  Some total functions need loops.
- A prim. rec. **program** computes a prim. rec. **function**.

Sound

Incomplete

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.
  Some total functions need loops.
- A prim. rec. **program** computes a prim. rec. **function**.

Sound

Incomplete

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.

  Some total functions need loops.

- A prim. rec. **program** computes a prim. rec. **function**.

  Each prim. rec. **function** is computed by a prim. rec. **program**.

Sound

Extensionally complete

Incomplete

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.
  Some total functions need loops.
- A prim. rec. **program** computes a prim. rec. **function**.
  Each prim. rec. **function** is computed by a prim. rec. **program**.
- A LOOP program computes a prim. rec. function.
  Each prim. rec. function is computed by a LOOP program.

Sound

Extensionally complete

Incomplete

Examples

Syntactic criterion

Semantic property

- A program without loops always terminate.
  Some total functions need loops.
- A prim. rec. **program** computes a prim. rec. **function**.
  Each prim. rec. **function** is computed by a prim. rec. **program**.
- A LOOP program computes a prim. rec. function.
  Each prim. rec. function is computed by a LOOP program.

Sound

Extensionally complete

Incomplete

Complexity of a Function

- Each program has a complexity.
- Each function is computed by several programs.
- The complexity of a function is the smallest complexity of programs computing it.

Complexity of a Function

- Each program has a complexity.
- Each function is computed by several programs.
- The complexity of a function is the smallest complexity of programs computing it.

Example (sorting):

- Insertion sort: $O(n^2)$.
- Quick sort: $O(n \log(n))$.
- Sorting function: $O(n \log(n))$.

Implicit Complexity

- Each program has a complexity.
- Each program computes one function.
- The complexity of the function may be smaller than the complexity of the program.

Implicit Complexity

- Each program has a complexity.
- Each program computes one function.
- The complexity of the function may be smaller than the complexity of the program.

Example (insertion sort):

- Insertion sort: $O(n^2)$, sorting function: $O(n \log(n))$.
- Explicit complexity: $O(n^2)$.
- Implicit complexity: $O(n \log(n))$.

Termination Orderings

First Order Constructors TRS

Three disjoint sets of function ($\mathtt{f} \in \mathcal{F}$), constructors ($\mathbf{c} \in \mathcal{C}$) and variables ($x \in \mathcal{V}$);

$$
\begin{array}{lll}
\textit{(Constructor terms)} & \mathcal{T}(\mathcal{C}) \ni u & ::= \mathbf{c} \mid \mathbf{c}(u_1, \cdots, u_n) \\
\textit{(terms)} & \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{V}) \ni t & ::= \mathbf{c} \mid x \mid \mathbf{c}(t_1, \cdots, t_n) \mid \\
 & & \quad\; \mathtt{f}(t_1, \cdots, t_n) \\
\textit{(patterns)} & \mathcal{P} \ni p & ::= \mathbf{c} \mid x \mid \mathbf{c}(p_1, \cdots, p_n) \\
\textit{(rules)} & \mathcal{D} \ni d & ::= \mathtt{f}(p_1, \cdots, p_n) \to t
\end{array}
$$

First Order Constructors TRS

Three disjoint sets of function ($\mathtt{f} \in \mathcal{F}$), constructors ($\mathbf{c} \in \mathcal{C}$) and variables ($x \in \mathcal{V}$);

$$
\begin{array}{lll}
\textit{(Constructor terms)} & \mathcal{T}(\mathcal{C}) \ni u & ::= \mathbf{c} \mid \mathbf{c}(u_1, \cdots, u_n) \\
\textit{(terms)} & \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{V}) \ni t & ::= \mathbf{c} \mid x \mid \mathbf{c}(t_1, \cdots, t_n) \mid \\
 & & \quad\ \mathtt{f}(t_1, \cdots, t_n) \\
\textit{(patterns)} & \mathcal{P} \ni p & ::= \mathbf{c} \mid x \mid \mathbf{c}(p_1, \cdots, p_n) \\
\textit{(rules)} & \mathcal{D} \ni d & ::= \mathtt{f}(p_1, \cdots, p_n) \to t
\end{array}
$$

No defined symbols in patterns.

First Order Constructors TRS

Three disjoint sets of function ($\mathtt{f} \in \mathcal{F}$), constructors ($\mathbf{c} \in \mathcal{C}$) and variables ($x \in \mathcal{V}$);

$$
\begin{array}{lll}
\textit{(Constructor terms)} & \mathcal{T}(\mathcal{C}) \ni u & ::= \mathbf{c} \mid \mathbf{c}(u_1, \cdots, u_n) \\
\textit{(terms)} & \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{V}) \ni t & ::= \mathbf{c} \mid x \mid \mathbf{c}(t_1, \cdots, t_n) \mid \\
 & & \qquad \mathtt{f}(t_1, \cdots, t_n) \\
\textit{(patterns)} & \mathcal{P} \ni p & ::= \mathbf{c} \mid x \mid \mathbf{c}(p_1, \cdots, p_n) \\
\textit{(rules)} & \mathcal{D} \ni d & ::= \mathtt{f}(p_1, \cdots, p_n) \to t
\end{array}
$$

No defined symbols in patterns.

A program is a set of rules with a main symbol.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Strict monotonicity: $t_i < t'_i$ implies $\mathtt{f}(\ldots, t_i, \ldots) < \mathtt{f}(\ldots, t'_i, \ldots)$.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \to r$, we have $l > r$ (for each substitution).

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \to r$, we have $l > r$ (for each substitution).

Theorem (Dershowitz)

A program with a termination ordering terminates uniformly.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \to r$, we have $l > r$ (for each substitution).

Theorem (Dershowitz)

A program with a termination ordering terminates uniformly.

Sketch of proof

$\mathtt{f}(\ldots, \text{redex}, \ldots)$ reduces to $\mathtt{f}(\ldots, \text{contractum}, \ldots)$.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \to r$, we have $l > r$ (for each substitution).

Theorem (Dershowitz)

A program with a termination ordering terminates uniformly.

Sketch of proof

$\mathtt{f}(\ldots, \text{redex}, \ldots)$ reduces to $\mathtt{f}(\ldots, \text{contractum}, \ldots)$.
redex $>$ contractum because the rules are ordered.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \to r$, we have $l > r$ (for each substitution).

Theorem (Dershowitz)

A program with a termination ordering terminates uniformly.

Sketch of proof

$\mathtt{f}(\ldots, \text{redex}, \ldots)$ reduces to $\mathtt{f}(\ldots, \text{contractum}, \ldots)$.
redex $>$ contractum because the rules are ordered.
$\mathtt{f}(\ldots, \text{redex}, \ldots) > \mathtt{f}(\ldots, \text{contractum}, \ldots)$ by monotonicity.

Termination Orderings

Ordering on terms, strictly monotonous and well-founded.

Definition

A program admit a termination ordering $>$ iff for each rule $l \rightarrow r$, we have $l > r$ (for each substitution).

Theorem (Dershowitz)

A program with a termination ordering terminates uniformly.

Sketch of proof

$\mathtt{f}(\ldots, \text{redex}, \ldots)$ reduces to $\mathtt{f}(\ldots, \text{contractum}, \ldots)$.
$\text{redex} > \text{contractum}$ because the rules are ordered.
$\mathtt{f}(\ldots, \text{redex}, \ldots) > \mathtt{f}(\ldots, \text{contractum}, \ldots)$ by monotonicity.
No infinite reduction by noetherianity.

Specific Termination Orderings

Lemma

For each uniformly terminating system, there exists a termination ordering.

Specific Termination Orderings

Lemma

For each uniformly terminating system, there exists a termination ordering.

Hint of proof

$t > s$ iff $t \xrightarrow{+} s$

Specific Termination Orderings

Lemma

For each uniformly terminating system, there exists a termination ordering.

Hint of proof

$t > s$ iff $t \xrightarrow{+} s$

- Compatible with the rules by construction.

- $l \to r$ implies $l > r$.

Specific Termination Orderings

Lemma

For each uniformly terminating system, there exists a termination ordering.

Hint of proof

$t > s$ iff $t \xrightarrow{+} s$

- Compatible with the rules by construction.
- Monotonic by definition of redex/contractum.

- $l \to r$ implies $l > r$.
- $\mathtt{f}(\ldots, \text{redex}, \ldots) \to \mathtt{f}(\ldots, \text{contractum}, \ldots)$.

Specific Termination Orderings

Lemma

For each uniformly terminating system, there exists a termination ordering.

Hint of proof

$t > s$ iff $t \xrightarrow{+} s$

- Compatible with the rules by construction.
- Monotonic by definition of redex/contractum.
- Well-founded … because the system terminates!

- $l \to r$ implies $l > r$.
- $\texttt{f}(\ldots, \text{redex}, \ldots) \to \texttt{f}(\ldots, \text{contractum}, \ldots)$.
- …

Generic Termination Orderings

- The existence of a termination ordering is undecidable.
- Having different orderings for each system is inconvenient.

Generic Termination Orderings

- The existence of a termination ordering is undecidable.
- Having different orderings for each system is inconvenient.

Problem

Find a termination ordering independently of the TRS but still be able to prove termination of many systems.

Generic Termination Orderings

- The existence of a termination ordering is undecidable.
- Having different orderings for each system is inconvenient.

Problem

Find a termination ordering independently of the TRS but still be able to prove termination of many systems.

Idea (RPO in a nutshell)

- If `f` calls `g` and `g` never calls `f`, then going from `f` to `g` is a step toward termination.
- During a recursive call, something must decrease inside the arguments.

Recursive Path Ordering

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \ldots, s_i, \ldots, s_m) = s$$

Recursive Path Ordering

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \ldots, s_i, \ldots, s_m) = s$$

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

Recursive Path Ordering

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \ldots, s_i, \ldots, s_m) = s$$

$<_{\mathcal{F}}$ ordering of $\mathcal{F} \cup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

Recursive Path Ordering

$$t = \mathtt{f}(t_1, \cdots, t_n) \prec_{rpo} \mathtt{g}(s_1, \ldots, s_i, \ldots, s_m) = s$$

$<_{\mathcal{F}}$ ordering of $\mathcal{F} \cup \mathcal{C}$.

$$\frac{\exists i, t \preceq_{rpo} s_i}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} \mathtt{g}(s_1, \cdots, s_m) \quad f <_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec^r_{rpo} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

MPO, LPO, PPO

Comparing arguments of recursive calls.

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec^{r}_{rpo} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

MPO, LPO, PPO

Comparing arguments of recursive calls.

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec^r_{rpo} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

- MPO: multiset ordering.
- LPO: lexicographic ordering.
- PPO: product ordering.

MPO, LPO, PPO

Comparing arguments of recursive calls.

$$\frac{\forall i, t_i \prec_{rpo} s \quad \{t_1, \cdots, t_n\} \prec^r_{rpo} \{s_1, \cdots, s_n\} \quad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}}{t \prec_{rpo} s}$$

- MPO: multiset ordering.
- LPO: lexicographic ordering.
- PPO: product ordering.

Exercise

Prove that they all are termination orderings. . .

Implicit Complexity

Theorem (Hofbauer, BMM)

$PPO \equiv MPO \equiv \textsc{PrimRec}.$

Implicit Complexity

Theorem (Hofbauer, BMM)

$PPO \equiv MPO \equiv \textsc{PrimRec}$.

- Systems terminating by PPO/MPO compute all the PrimRec functions (extensional completeness, easy).
- Systems terminating by PPO/MPO compute only the PrimRec functions (soundness, hard).
- No intensional completeness (quick sort).

Implicit Complexity

Theorem (Hofbauer, BMM)

$PPO \equiv MPO \equiv$ PrimRec.

- Systems terminating by PPO/MPO compute all the PrimRec functions (extensional completeness, easy).
- Systems terminating by PPO/MPO compute only the PrimRec functions (soundness, hard).
- No intensional completeness (quick sort).

Theorem (Weierman)

$LPO \equiv$ *Multiple recursive functions.*

Interpretations

Principle of Interpretations

Termination orderings are powerful but hard to invent (prove noetherianity).
Idea: instead of trying to build orders on terms (complicated structure), try to interpret terms in a well known ordered set.

$$[\![\bullet]\!] : \mathcal{T} \to (A, <) \text{ and then, } t \prec s \text{ iff } [\![t]\!] < [\![s]\!]$$

Principle of Interpretations

Termination orderings are powerful but hard to invent (prove noetherianity).
Idea: instead of trying to build orders on terms (complicated structure), try to interpret terms in a well known ordered set.

$$[\![\bullet]\!] : \mathcal{T} \to (A, <) \text{ and then, } t \prec s \text{ iff } [\![t]\!] < [\![s]\!]$$

- $(A, <)$ is well founded.
- $[\![t]\!] > [\![t']\!]$ implies $[\![\mathtt{f}(\ldots, t, \ldots)]\!] > [\![\mathtt{f}(\ldots, t', \ldots)]\!]$.
- For each rule $l \to r$, we have $[\![l]\!] > [\![r]\!]$.

Principle of Interpretations

Termination orderings are powerful but hard to invent (prove noetherianity).
Idea: instead of trying to build orders on terms (complicated structure), try to interpret terms in a well known ordered set.

$$[\![\bullet]\!] : \mathcal{T} \to (A, <) \text{ and then, } t \prec s \text{ iff } [\![t]\!] < [\![s]\!]$$

- $(A, <)$ is well founded.
- $[\![t]\!] > [\![t']\!]$ implies $[\![\mathtt{f}(\dots, t, \dots)]\!] > [\![\mathtt{f}(\dots, t', \dots)]\!]$.
- For each rule $l \to r$, we have $[\![l]\!] > [\![r]\!]$.

Easy

Principle of Interpretations

Termination orderings are powerful but hard to invent (prove noetherianity).
Idea: instead of trying to build orders on terms (complicated structure), try to interpret terms in a well known ordered set.

$$[\![\bullet]\!] : \mathcal{T} \to (A, <) \text{ and then, } t \prec s \text{ iff } [\![t]\!] < [\![s]\!]$$

- $(A, <)$ is well founded.
- $[\![t]\!] > [\![t']\!]$ implies $[\![\mathtt{f}(\dots, t, \dots)]\!] > [\![\mathtt{f}(\dots, t', \dots)]\!]$.
- For each rule $l \to r$, we have $[\![l]\!] > [\![r]\!]$.

Easy Use compositionality

Principle of Interpretations

Termination orderings are powerful but hard to invent (prove noetherianity).
Idea: instead of trying to build orders on terms (complicated structure), try to interpret terms in a well known ordered set.

$$[\![\bullet]\!] : \mathcal{T} \to (A, <) \text{ and then, } t \prec s \text{ iff } [\![t]\!] < [\![s]\!]$$

- $(A, <)$ is well founded.
- $[\![t]\!] > [\![t']\!]$ implies $[\![\mathtt{f}(\ldots, t, \ldots)]\!] > [\![\mathtt{f}(\ldots, t', \ldots)]\!]$.
- For each rule $l \to r$, we have $[\![l]\!] > [\![r]\!]$.

Easy Use compositionality Hard

Compositional Interpretations

- For each symbol f of arity n, define a function $[\![f]\!] : A^n \to A$.
- Extend recursively $[\![f(t_1, \cdots, t_n)]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.
- Define $t \prec s$ iff $[\![t]\!] < [\![s]\!]$.

Compositional Interpretations

- For each symbol f of arity n, define a function $[\![f]\!] : A^n \to A$.
- Extend recursively $[\![f(t_1, \cdots, t_n)]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.
- Define $t \prec s$ iff $[\![t]\!] < [\![s]\!]$.

$[\![f]\!]$ has subterm property if $[\![f]\!](X_1, \cdots, X_n) \geq X_i$.

Lemma

If each $[\![f]\!]$ is monotonic and has subterm property, then $\prec$ is monotonic.

Compositional Interpretations

- For each symbol f of arity n, define a function $[\![f]\!] : A^n \to A$.
- Extend recursively $[\![f(t_1, \cdots, t_n)]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.
- Define $t \prec s$ iff $[\![t]\!] < [\![s]\!]$.

$[\![f]\!]$ has subterm property if $[\![f]\!](X_1, \cdots, X_n) \geq X_i$.

Lemma

If each $[\![f]\!]$ is monotonic and has subterm property, then $\prec$ is monotonic.

Ackermann

Ackermann's function admit an interpretation over the ordinal numbers.

Polynomial Interpretations

- Polynomial interpretation: $[\![f]\!](X_1, \cdots, X_n)$ is a polynomial (with positive integer coefficients).
- A TRS admits a polynomial interpretation if $[\![l]\!] > [\![r]\!]$. It defines a termination ordering.
- A TRS admitting a polynomial interpretation terminates uniformly.

Polynomial Interpretations

- Polynomial interpretation: $[\![f]\!](X_1, \cdots, X_n)$ is a polynomial (with positive integer coefficients).
- A TRS admits a polynomial interpretation if $[\![l]\!] > [\![r]\!]$. It defines a termination ordering.
- A TRS admitting a polynomial interpretation terminates uniformly.

Exponential

$$\begin{array}{rcl} \mathtt{db}(\mathbf{z}) & \to & \mathbf{z} \\ \mathtt{db}(\mathbf{S}(x)) & \to & \mathbf{S}'(\mathbf{S}'(\mathtt{db}(x))) \\ \mathtt{exp}(\mathbf{z}) & \to & \mathbf{S}(\mathbf{z}) \\ \mathtt{exp}(\mathbf{S}(x)) & \to & \mathtt{db}(\mathtt{exp}(x)) \end{array}$$

$[\![\mathbf{z}]\!] = 1 \qquad [\![\mathbf{S}]\!](X) = 2X + 4 \qquad [\![\mathbf{S}']\!](X) = X + 1$

$[\![\mathtt{db}]\!](X) = 2X + 1 \qquad [\![\mathtt{exp}]\!](X) = X + 2$

Polynomial Interpretations

Theorem (BCMT)

The TRS admitting a polynomial interpretation characterize EXP2TIME.

Polynomial Interpretations

Theorem (BCMT)

The TRS admitting a polynomial interpretation characterize EXP2TIME.

Where does the exponential come from?

$[\![\mathbf{S}]\!](X) = 2X$ hence $[\![\mathbf{S}^n(\mathbf{z})]\!] = 2^n$

$[\![\mathbf{S}]\!](X) = X^3, [\![\mathbf{z}]\!] = 2$ hence $[\![\mathbf{S}^n(\mathbf{z})]\!] = 2^{3^n}$

Polynomial Interpretations

Theorem (BCMT)

The TRS admitting a polynomial interpretation characterize EXP2TIME.

Where does the exponential come from?

$[\![\mathbf{S}]\!](X) = 2X$ hence $[\![\mathbf{S}^n(\mathbf{z})]\!] = 2^n$

$[\![\mathbf{S}]\!](X) = X^3, [\![\mathbf{z}]\!] = 2$ hence $[\![\mathbf{S}^n(\mathbf{z})]\!] = 2^{3^n}$

Observation

The interpretation of constructors is crucial for complexity.

Interpretations of Constructors

The interpretation of a symbol is

- Additive: $[\![f]\!](X_1, \cdots, X_n) = \sum X_i + a$
- Multiplicative: $[\![f]\!](X_1, \cdots, X_n)$ has degree 1.
- Polynomial: $[\![f]\!](X_1, \cdots, X_n)$ is any polynomial.

Interpretations of Constructors

The interpretation of a symbol is

- Additive: $[\![f]\!](X_1, \cdots, X_n) = \sum X_i + a$
- Multiplicative: $[\![f]\!](X_1, \cdots, X_n)$ has degree 1.
- Polynomial: $[\![f]\!](X_1, \cdots, X_n)$ is any polynomial.

Theorem (BCMT)

Depending on the interpretation of constructors, the TRS admitting a polynomial interpretation characterize:

- *Additive* $\Rightarrow$ PTIME.
- *Multiplicative* $\Rightarrow$ EXPTIME.
- *Polynomial* $\Rightarrow$ EXP2TIME.

Interpretations are too large

Smallest polynomial interpretation for addition?

$$\begin{array}{rcl} \mathtt{add}(\mathbf{z}, y) & \to & y \\ \mathtt{add}(\mathbf{S}(x), y) & \to & \mathbf{S}(\mathtt{add}(x, y)) \end{array}$$

Interpretations are too large

Smallest polynomial interpretation for addition?

$$\begin{array}{rcl} \texttt{add}(\mathbf{z}, y) & \to & y \\ \texttt{add}(\mathbf{S}(x), y) & \to & \mathbf{S}(\texttt{add}(x, y)) \end{array}$$

$[\![\mathbf{z}]\!] = 1 \quad [\![\mathbf{S}]\!](X) = X + 1 \qquad [\![\texttt{add}]\!](X, Y) = 2X + Y$

Interpretations are too large

Smallest polynomial interpretation for addition?

$$\begin{array}{rcl} \mathtt{add}(\mathbf{z}, y) & \to & y \\ \mathtt{add}(\mathbf{S}(x), y) & \to & \mathbf{S}(\mathtt{add}(x, y)) \end{array}$$

$[\![\mathbf{z}]\!] = 1 \quad [\![\mathbf{S}]\!](X) = X + 1 \qquad [\![\mathtt{add}]\!](X, Y) = 2X + Y$

If we take the more natural $[\![\mathtt{add}]\!](X, Y) = X + Y$, then the second rule is not strictly decreasing:
$[\![\mathtt{add}(\mathbf{S}(x), y)]\!] = X + Y + 1 = [\![\mathbf{S}(\mathtt{add}(x, y))]\!]$

Quasi Interpretation

- We relax the condition on rules: $(\!|l|\!) \geq (\!|r|\!)$.
- Termination is not assured: $\mathtt{f}(x) \to \mathtt{f}(x)$. We need an extra termination proof.

Quasi Interpretation

- We relax the condition on rules: $(\!|l|\!) \geq (\!|r|\!)$.
- Termination is not assured: $\mathtt{f}(x) \to \mathtt{f}(x)$. We need an extra termination proof.
- But a size bound is still assured: $t \xrightarrow{!} v$ implies $(\!|t|\!) \geq (\!|v|\!)$.
- Hence, if $(\!|t|\!)$ is polynomial (in the inputs), all the value handled during reduction also have polynomial size.

Quasi Interpretation

- We relax the condition on rules: $(\!|l|\!) \geq (\!|r|\!)$.
- Termination is not assured: $\mathtt{f}(x) \to \mathtt{f}(x)$. We need an extra termination proof.
- But a size bound is still assured: $t \xrightarrow{!} v$ implies $(\!|t|\!) \geq (\!|v|\!)$.
- Hence, if $(\!|t|\!)$ is polynomial (in the inputs), all the value handled during reduction also have polynomial size.

Theorem (BMM)

$MPO{+}QI \equiv$ PTIME

TRS terminating by MPO and admitting an additive QI characterize PTIME.

Example: Longest Common Subsequence

$$
\begin{array}{rcl}
\mathtt{lcs}(x, \epsilon) & \rightarrow & \mathbf{z} \\
\mathtt{lcs}(\epsilon, y) & \rightarrow & \mathbf{z} \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) & \rightarrow & \mathbf{S}(\mathtt{lcs}(x, y)) \\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) & \rightarrow & \mathtt{max}(\mathtt{lcs}(x, \mathbf{j}(y)), \mathtt{lcs}(\mathbf{i}(x), y))
\end{array}
$$

$(\!|lcs|\!)(X, Y) = (\!|max|\!)(X, Y) = \max(X, Y)$ No interpretation.

Example: Longest Common Subsequence

$$\begin{array}{rcl} \mathtt{lcs}(x,\epsilon) & \to & \mathbf{z} \\ \mathtt{lcs}(\epsilon,y) & \to & \mathbf{z} \\ \mathtt{lcs}(\mathbf{i}(x),\mathbf{i}(y)) & \to & \mathbf{S}(\mathtt{lcs}(x,y)) \\ \mathtt{lcs}(\mathbf{i}(x),\mathbf{j}(y)) & \to & \mathtt{max}(\mathtt{lcs}(x,\mathbf{j}(y)),\mathtt{lcs}(\mathbf{i}(x),y)) \end{array}$$

$(\!|lcs|\!)(X,Y) = (\!|max|\!)(X,Y) = \max(X,Y)$ No interpretation.

- Explicit complexity: $O(2^n)$.
- Implicit complexity: $O(n^2)$.
- We can use memoisation (automated dynamic programming) to transform the program and reach the good complexity.
- Better expressivity than interpretations, but the method is far from intensional completeness (divide and conquer algorithms).

Conclusion

Conclusion

- Implicit computational complexity: syntactic criterions for semantic properties.
- Dream usage: certified compilation, proof carrying code.
- Proofs are hard but many results have been obtained in the past 20 years.
- Interpretation methods give a guideline for finding new characterizations.
- Interpretations are not restricted to TRS.
- Getting close to intensional completeness is extremely hard.